



Lattice Works

Spring 1988

Lattice® News

Lattice To Release New Document Composition Package

In the next few months, Lattice will be releasing **HighStyle™**, our new Document Composition System. The new package is ideal for creating software manuals, catalogs, or other large, complex documents. It is also well-suited for producing memos and letters, proposals, or any document which frequently changes content but rarely changes format.

HighStyle controls page design through more than 100 straight forward, "English-like" commands and functions which are embedded in the document text files or maintained in separate "format" files. Pre-defined formats and designs can be inserted into a document as it is being formatted, so the document's content is kept completely separate from its design. This also ensures that memos, letters, etc. appear exactly the same way each time they are printed regardless of who writes them.

Wayne Nartker, Lattice's Vice President of Sales and Marketing points out that **HighStyle's** major strengths are its text-handling abilities compared to *Paste-Up* Desktop Publishing Systems.

Continued on next page...

This chart shows some of HighStyle's special abilities not found in other desktop publishing packages. If these features are important to you, call Lattice for more information today.

Inside:

Lattice News

See pages 1-3

- New Document Composition Package
- Trade Show Drawing Winners

Focus on C

See pages 3-9

- New Lattice C Soon
- Training Classes Available
- Lattice Views Industry Topics
- Technical Q & A

Focus on Amiga

See pages 9-12, 16

- Undocumented Features of Lattice AmigaDOS C Compiler, Version 4.01
- "Compiler Companion" Available
- Technical Q & A

Focus on RPG

See pages 12-14

- New Programs Strengthen RPG II Support
- Technical Q & A

Current Product Version Numbers

See page 15

Feature	HighStyle	PageMaker	Xerox Ventura
Manual kerning	Yes	Yes	Yes
Automatic paragraph leading	Yes	Yes	Yes
Single word underline	Yes	No	Yes
Automatic control of widows	Yes	No	Yes
Automatic creation of tables	Yes	No	No
Automatic numbering of lists	Yes	No	Yes
Automatic "bullet" points	Yes	No	Yes
Automatic Keyword points	Yes	No	No
Automatic table of contents	Yes	No	Yes
Automatic indexing	Yes	No	Yes
Allows extensive text revision	Yes	No	Yes
Graphics card/display required	No	Yes	Yes

HighStyle

...Continued from previous page

"With **HighStyle**, it is not necessary to reformat a design when text is added or deleted from a document. The design of any document can be completely altered by changing only one or two lines, rather than the tedious process of re-formatting the entire document."

While **HighStyle** focuses its power on text formatting, it does not neglect graphics capabilities. You can create or alter graphic symbols with the package's icon editor. Any screen image can be captured with the "Snapshot" utility and called into the document as the document is being printed. **HighStyle** will create charts, tables, and graphs in far less time than any other system we've seen, and it will even generate bar codes.

HighStyle will work on almost every computer system without requiring hardware, memory, or operating system upgrades to work properly. It runs under DOS 2.1 and above. You do not need a mouse or other "pointer device"; a graphics card is not required; and **HighStyle** works just fine without other programs such as *Windows* or *GEM*. (You will need a graphics card to use the icon editor.)

Unlike many desktop publishing systems, **HighStyle** is very easy to use for a beginner. At its simplest level, anyone can take the software out of the box and begin creating highly-polished documents in a matter of minutes. Changing the type size and style is as easy as inserting commands as simple as "begin largertext," "begin smalltext," etc.. **HighStyle** assumes automatic control of design elements such as headers, footers, page numbering, line spacing,

columns, pagination, and hyphenation. However, with simple commands, you can permanently set or temporarily change these settings as well as the program defaults for kerning, leading, justification and other typesetting factors.

HighStyle's power, however, comes from its complete programmability. Documents can contain an unlimited number of macro commands. You can even "nest" macros and conditional commands to reduce complex functions into single commands. You can also re-program all the commands and macros provided in the package to completely customize your system. Once a document's design is established, conditional commands insure a consistent appearance of a frequently used format regardless of the document's content or length.

Lattice has been using **HighStyle** internally for almost a year now. All our new software manuals are being created using this system. Hewlett-Packard has also used the product to create the manuals included with their LaserJet printers. Now, Lattice will begin sharing the product with others. Based on our experiences, we will also offer training classes to help users take full advantage of **HighStyle's** substantial features.

The retail price of **HighStyle** will be \$200. The complete package will include a word processor, type font manager, icon editor, screen image "snapshot" processor, screen previewer, and formatter. Also included are several icons, pre-defined forms, and examples.

HighStyle will be available through Lattice dealers and distributors worldwide or directly from Lattice. For further information on **HighStyle**, contact Lattice Sales at 2500 South Highland Avenue, Lombard, IL 60148 or call us at (312) 916-1600.

Lattice Works is published by Lattice, Incorporated. We would like to include news from and about users. Please send comments, suggestions, and news stories to our attention. Thank You.

Lattice Works will be sent to all of our registered customers. Please be sure to complete and return the registration card enclosed with your purchase.

Trade Show Drawings Winner

We'd like to thank each of you who stopped by our booth at the various trade shows

Lattice attended last year. We're always glad to get the chance to meet our customers and find out from you what you want to see in Lattice products. We hope you were able to find out from us more about the wide range of products we offer and what you can expect in coming years.

As you may remember, at each show, we were asking you to enter your names for drawings to be held for free products. Winners have already been contacted. They are:

Info '87:

SecretDisk II

Matthew Lampell, CPCU
Western Union Telegraph Co.
One Lake Street
Upper Saddle River, NJ 07458

Focus on C

Two New Versions on the Horizon

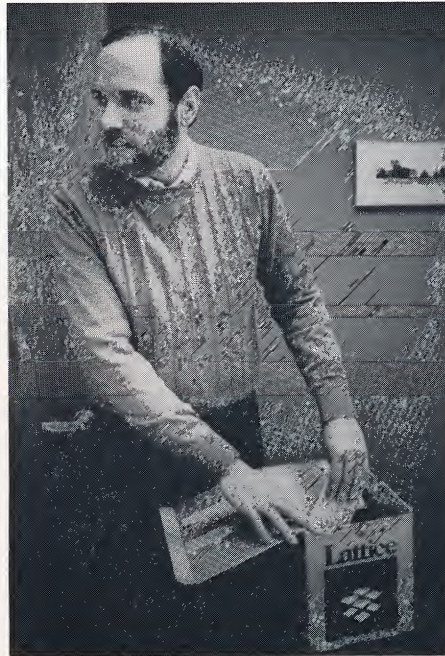
Let anyone think that Lattice is losing ground in the C Compiler battles, we want to reassure you now that the opposite is true. We have been devoting a huge amount of R&D into upgrading our current MS-DOS C compiler and creating an all new compiler as well.

Within weeks, all registered users of a Lattice C Compiler will receive a notice in the mail of our latest version, 3.3. The new version of the compiler runs under and generates code for both MS-DOS and OS/2. In addition, versions of the Lattice Screen Editor (LSE) and the C-SPRITE debugger which also run under both OS/2 as well as MS-DOS are now being bundled with the compiler.



SideTalk

Roger Stone
Utah Valley Community College
400 Catalpa, #302A
Midvale, UT 84047



Dan Knopoff selects winners from those of you who registered for free product drawings. We look forward to seeing you at our booths again at the trade shows where we will exhibit in 1988.

AmigaDOS C Development System

Douglas Campbell
Hebco Electronics Inc.
8939 Edwards Road
Cincinnati, OH 45209

RPG II Development System

J. Eatherly
DRC
1828 L Street, NW
Washington, DC 20036

3X Show, Washington, DC:

RPG II Development System

Kenneth Berlin
American Society for Training & Dev.
1630 Duke Street, Box 1443
Alexandria, VA 22313

Based on the attention Lattice received at these past shows, we are planning to exhibit at several trade shows in 1988. We'll announce our trade show schedule in an upcoming issue of *Lattice Works* and look forward to seeing you there.

RPG II Development System

Diana Nischan
Supervisor of Systems Programming
11 Grammes Road
Allentown, PA 18103

AmiExpo, New York:

AmigaDOS C Development System

Harry Lewis, President
Blue Bell Design, Inc.
524 White Oak Road
Blue Bell, PA 19422

COMDEX/Fall:

SecretDisk II

Kyle Keenia
NI Industries
5215 W. Boyle Avenue
Los Angeles, CA 90039

s of Lattice C

Other features of Version 3.3 include compliance with emerging ANSI C standards, improved embedded system support, enhancements to the standard libraries, as well as bug fixes and other advances in the compiler. The price also has been improved with this latest version's suggested retail price being lowered to \$450.00. If you are a registered user of the Lattice MS-DOS C Compiler, you will be able to upgrade for only \$75.00.

Lattice President, Dave Schmitt stated, "This new version provides a convenient path for a transition from MS-DOS to OS/2. You can use this product on an MS-DOS system to create programs that run under OS/2 protected

mode, or you can run the compiler in OS/2 protected mode to create programs that run under MS-DOS. The versions of the compiler, LSE and C-SPRITE run under either OS/2 or MS-DOS. A "switch" on the compiler determines whether it will generate code for MS-DOS, OS/2, or "family mode" programs which run under both.

"We have made a strong commitment to our C Compiler and C programming tools," added Wayne Nartker. "In terms of features and value, this version keeps the Lattice C Compiler even with the competition. Then, within just a few months of releasing Version 3.3, we will release an entirely new product based on the Lattice C Compiler. We are calling this new product Lattice C, Version 4.0. When it is released, we will treat it as a brand new product because it represents so many new features that calling it another upgrade won't do it justice. (*Don't worry if you have the current C Compiler;*

you will be able to obtain this new product through the update service)

Continued on next page...

"The new package will provide a complete programming environment consisting of the best tools available."



Lattice C Versions

...Continued from previous page

Those of you who had the opportunity to visit Lattice during the COMDEX show in November had a chance to preview portions of this all new C compiler which will be released this summer.

Currently completing development, Version 4.0's major new features include an integrated source level debugger, Lattice's own true overlay linker, built-in functions, new keywords, and a new "HUGE" memory model. In addition, Version 4.0 offers further compliance with ANSI standards, improved floating point arithmetic, and numerous optimizations which result in the compiler generating smaller, faster code.

Dave Schmitt explains, "You know how the competition among C language vendors has been increasing. We are convinced that this complete remake of the Lattice C Compiler will prove to you that Lattice is in this market — not only to stay — but to be the best C Compiler available."

"Lattice will also offer programmers the best value," points out Wayne Nartker. "The new package will provide a complete programming environment consisting of the best tools available. We will be including our new global optimizer, debugger, an overlay linker that will optionally overlay data, our "Make" utility, and an enhanced Cross Reference Generator. Plus we are adding our file handling utilities including GREP, DIFF, EXTRACT, BUILD, WC, SPLAT, and FILES. Finally, we are adding the Lattice Screen Editor. Like Lattice C Version 3.3, the compiler as well as each of the utilities will run under OS/2 and MS-DOS.

"These are not flimsy or stripped down tools that we have just thrown together," Wayne said. "The Make utility, overlay linker, and debugger offer features not available in any other company's products. The text editor and file handling utilities are the same full-featured products Lattice has offered separately in the past. Yet now we are including them with the compiler at no extra cost.

Along with the increased value of the

product and the better code generation, one of the most most important changes in the new product is the new Source Level debugger which will be included in the compiler package. We know how many of you like our competitor's Code View debugger, so we incorporated many of Code View's features into this product plus added a few of our own. If you saw the new debugger at COMDEX, you saw how programs can be debugged by displaying source code, assembly code or both. You can single step by C source line or by machine instruction. Register windows allow you to watch CPU registers and flags as the program executes. The new debugger also displays values of variables, addresses, or ranges of memory. Its full set of machine level commands allow users to debug at the assembly level as well as the C source level.

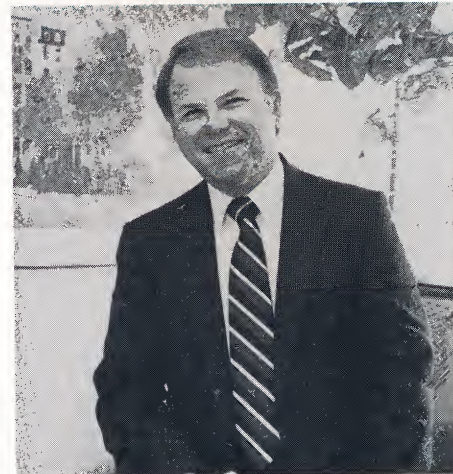
The new debugger runs in full screen, multiple window mode or in command line mode. It automatically detects a PC's video hardware to support all common color and monochrome monitors. Dual monitor support keeps debugger output separate from the program output, and a screen swap feature ensures this separation on single monitor systems. It also features on-line help with pull-down menus and function key commands. A restart command will let you restart the program with new command line arguments without exiting the debugger. Its use of virtual memory allows both low memory usage and high speed.

The new C Compiler introduces the "HUGE" keyword and Huge memory model. Using the "HUGE" keyword allows "FAR" pointers to use more efficient code generation techniques which result in smaller .EXE files. The Huge memory model allows programs to exceed the 64K static data limit without requiring "FAR" or "HUGE" keywords.

Other new keywords in Version 4.0 include "CHIP" (which replaces Lattice's previous "CONST" keyword) and the ANSI keywords "CONST" and "VOLATILE." Using "CONST" allows the compiler to perform certain optimizations while "VOLATILE" prevents the compiler from performing optimizations that could be destructive to the application.

To increase speed and generate smaller code, the new compiler incorporates

"...the new Lattice C Compiler will prove to everyone that Lattice is in this market not only to stay, but to be the best C Compiler available."



many optimizations which include in-line functions that eliminate a number of library routines, and more efficient handling of Large memory model pointers. It also adds new ANSI library functions and ANSI pre-processor support.

"With all the changes in this product we have gone to great lengths to retain compatibility with earlier versions of the Lattice MS-DOS C Compiler," said Dave Schmitt. "In addition, this new version provides a convenient path for a transition from MS-DOS to OS/2. Although OS/2 requires changes in the way the compiler generates code, Lattice C Compilers will offer source level compatibility."

"In terms of flexibility, features, and value, our new Version 4.0 Compiler puts Lattice C ahead of the competition once again," said Wayne Nartker. "And as a current user, you know Lattice goes on to offer superior product support through the Lattice Bulletin Board, Byte's BIX network, Lattice Telephone Support Hotline, plus its frequent contact and notices sent to registered users. In addition, because Lattice C is available on a wide range of systems, you will find it much easier if you decide to move your applications to other machines."

As always, we will notify each of our registered users of the procedures to obtain the new compiler as soon as it is available. Watch your mail for notification.

Lattice Offers Training for C and OS/2

If you are deciding to learn the C language; If you will be creating programs to run under OS/2; Or if you will be responsible for implementing OS/2 in your company, you can take advantage of Lattice's extensive experience by attending our C Programming seminar or one of our two OS/2 seminars.

The C Programming classes provide an equivalent of a college level C programming course in four days of intense lecture and hands-on training. The course is designed for programmers who wish to learn the C language quickly. Knowledge of C, MS-DOS, or OS/2 is not required; however a basic knowledge of data processing is required, and knowledge of other programming languages would be helpful.

Our one-day seminar, "OS/2 For Managers", is designed for those of you who will be introducing OS/2 into your businesses as well as for those who must manage OS/2 product development. The three-day seminar, "OS/2 Programming" gives you an in-depth description of the OS/2 programming environment. All Lattice seminars are divided into lecture sessions and hands-on workshops.

Seminar Dates Scheduled at this time are:

C Programming	OS/2 For Managers	OS/2 Programming
May 3-6	April 28	April 19-21
May 24-27	May 19	May 10-12
June 21-24	June 30	June 14-16

The seminars are conducted by Lattice's technical and educational staff. Lattice created the original Lattice C Compiler in 1981, and we have been working with IBM on OS/2 since June 1986 — testing the new operating system and converting the Lattice C Compiler and programming tools. This experience has been distilled into three fast-paced, informative courses designed to bring programmers and managers "up-to-speed" on the C language and OS/2 as quickly as possible.

The C Programming class covers the difference between C and other languages, C source file structure, types and classes of variables, variable math, flow control, data types, branching, and arrays. Programming techniques examine variable logistics, function calls, function definitions, top-down design, arguments, file I/O, pointers, function pointers, as well as other I/O functions. In addition, the classes cover conditional compilations, recursive programming, structures and unions, and linked lists. The cost for this four-day seminar is \$895.00.

OS/2 For Managers examines the history and evolution of OS/2; how OS/2 compares to DOS and UNIX; Session Process, Memory, and File management; Inter-process Communication; Dynamic Linking; High-performance Human Interfaces; the DOS Compatibility Mode;

and Software Development Facilities. The future of OS/2 is examined — including the Presentation Manager, Extended Edition, OS/2 and SAA, and 80386 and future processors. The cost for this one-day seminar is \$395.00.

OS/2 Programming is a three-day session that examines the OS/2 architecture and feature set in detail; the Application Program Interface; Programming Design; and Programming Tools. Approximately half of the seminar involves hands-on work which includes several programming exercises to provide direct experience with the API and other aspects of OS/2 programming. The cost for the three-day seminar is \$895.00.

Classes are held in Lattice's training facilities in Lombard from 9:00 a.m. until 4:00 p.m.. Instructors will remain available after the seminars each day to answer questions or provide individual consultation. The seminar fee includes a continental breakfast, catered lunch, a comprehensive seminar workbook, and a set of OS/2 programmer productivity tools. Reservations may be made at any time up to two weeks prior to the seminar.

For further information or to register, contact: Education Department, Lattice, Inc., 2500 South Highland Avenue, Lombard, IL 60148. Telephone (312) 916-1600, extension 900.

Lattice Point of View

Questions, Comments & Answers

The following questions and issues have appeared on the Lattice Bulletin Board. If you have any questions on Lattice's position on an industry issue, please feel to post the question on our Bulletin Board or send the question to *Lattice Works*.

Null Pointers

Q. From: LARRY TYMPANICK
I have a problem with Lattice's defini-

Continued on next page...



Lattice's training classes give you a fast, efficient way to learn OS/2 and C Programming. Discover new tips and techniques from the experts at Lattice. Call our Education Department today.

Lattice Point of View

...Continued from previous page

tion of NULL. SPTR is unnecessary in my opinion, and I'm not even sure if it would work as is on another machine. Why don't you just declare NULL as (char *)0 ? My interpretation of NULL is that it should not be used as a replacement for int 0.

C. From: MIKE MILLER

K&R page 192, paragraph 7.14 states: "However, it is guaranteed that assignment of the constant 0 to a pointer will produce a null pointer distinguishable from a pointer to any object."

ANSI Draft (10/1/86) states: "A integral constant expression with the value 0, or such an expression cast to type void * is called a null pointer constant."

A NULL is a pointer with a value of zero.

C. From: GLENN DOBRATZ

Yes, there is a difference between a null pointer and a pointer to a void. But, I think the real question is: "Is a null pointer to a char different from a null pointer to a void?"

A purist would probably answer yes, and I think I would agree. ANSI says no, however, and right, wrong, or indifferent, they have the last say.

But isn't that really why #DEFINE exists? If you #define NULL and say that it will be used to represent pointers with null values, then does the mechanism used in the #define really matter? I don't think so.

I think this has digressed from the original question, which was, "Is there a better mechanism than SPTR & Co. for communicating pointer size in those very few instances where it really does matter whether pointers are 2 bytes long or four?" My answer was yes.

I proposed #define NULL (void *)0 as a viable alternative to SPTR. Since even a purist should object to using 0 or 0L for a null pointer, nothing short of having a bunch of NULLx's defined as (char *)0, (int *)0, (long *)0, . . . , should do. This is obviously too cumbersome to be really practical, so some compromise is needed. No definition of NULL is going to please everyone, but

at least (void *)0 avoids the undesirable situation where certain symbols are predefined by the compiler and the programmer may not have access to that symbol for their own purposes.

A. From: JOHN MEISSEN
(Lattice)

Prototyping was the original reason for defining NULL as 0 or 0L:

According to K&R, it is always permitted to compare a pointer to the integer constant '0'. Thus, regardless of the type of pointer a comparison to NULL when NULL is defined as 0 is always acceptable.

A. From: CARLOTTA
DECONCILIS (Lattice)

A NULL pointer is not a pointer which points to a string containing '0', 0L, '0'. It is a pointer which can be used to signify an invalid pointer — for example, a pointer which does not point to a valid string.

For functions such as strtok() and stptok(), a NULL pointer is returned. Does this mean that you can condition a break by saying: if (ptr == (char *) '0') ? It does not because by doing so, you are testing for a pointer to the null character which is not the same as a NULL pointer.

Optimized Code Generation

Q. From: CHARLES HALL

I have a question concerning the code generation for the Lattice 'C' compiler. I noticed this one day whilst debugging a function:

```
int n, cond;
.
.
while (1)
.
.
if (!n - cond) /* this statement */
break;
.
.
```

The above statement generates the following code (labels added):

```
cmp    word ptr [bp+4], 0    ;'n' is stored at
[bp+4]
jz     A1
xor    ax, ax
jmp    A2
A1:
xor    ax, ax
inc    ax
A2:
dec    word ptr [bp+4]
test   ax, ax
jz     TEST COND
jmp    END OF LOOP
TESTCOND:
.
.
```

This doesn't strike me as horribly efficient.

Isn't it possible for the compiler to generate better code than this? Since 'n' is an integer (i.e. one word), I would expect it to generate this:

```
mov    ax, word ptr [bp+4]
dec    word ptr [bp+4]
test   ax, ax
jnz    TEST COND
JMP    END OF LOOP
TEST COND:
.
.
```

Perhaps I expect too much! If the code generation is like this for this simple fragment, I wonder what it is like for other more complex expressions.

A. From: TOM PRODEHL
(Lattice)

It's always easy to find instances in which a compiler—any compiler—produces what seems to be less than obvious code. To look at a fragment of C source and its translation and say "Why didn't it do it this way" is not fair because you can spot the obvious solution to an isolated example but the compiler has to process all possible if statements, which means that it has to take a general approach. This means that in some cases less efficient code can be generated. To suppose that the problem only gets worse as the expressions get more complicated is false.

Also, it seems a little unreasonable to suggest that just because some code is legal C it should automatically generate the most efficient code. Optimizations, when put into a compiler have a cost/benefit ratio associated; rarely used con-

structs are the last to be optimized.

Perhaps one reason that the Lattice C compiler code generation is so general is that it is basically a portable compiler. Its original goal was not to be just a 80x86 compiler. The same basic compiler is on the Amiga, Atari ST, and is a cross compiler from others to 80x86. So code generation is abstracted to nearly the last step of object file generation.

Realize too that optimizations rest on ideology as well as science. Differing views on levels of concern for aliasing, the detection mechanisms for common subexpressions and so on can impact the code generated. Some optimizations can be 95% successful and not break the program but in 5% of the cases, you can discover that a "bug" goes away when the optimizer is turned off.

Product Support

Questions, Comments & Answers

The following questions and answers are taken from the Lattice Bulletin Board. The Lattice Bulletin Board is your best source of support for Lattice products, because in addition to the Lattice staff, many users of Lattice products are also willing and able to provide prompt answers. If you have any questions on any Lattice product, please feel free to write, call, send a message on our FAX, or utilize our Bulletin Board.

Running Out of Space on the Command Line

Q. From: GARRY SILVEY

Is it possible to specify a file name that contains the command line parameters for `lc`? I.E. Instead of entering "`LC -s xx yy zz`", can I enter "`LC -filename`", where the file named contains the above command line.

The reason for this is that I've run out of space in the command line, and have to keep shortening my file names to get `LC` to read the whole command line. It would be nice to put the command line in multiple lines in a data file, and have `LC` read that file.

A. From: JOHN RILEY (Lattice)

One answer is to put your source files in a subdir and do a "`LC *`" from inside that subdir. That way you don't have to type out all those file names.

A. From: CARL KUCK

You could also try using a "make" utility. Make would `exec()` the passes of the compiler, without using `LC.EXE`.

Ed. — Lattice also offers an excellent Make utility, called LMK.

A. From: JOHN RILEY (Lattice)

MS-DOS limits you to a 127 character command line. If you have THAT many -d options on your command line you should put those in a .H file.

SideTalk on a Network

Q. Can anyone tell me how to run SideTalk in foreground mode AFTER installing Novell Netware. Apparently the interrupts aren't compatible.

A. From: MIKE MILLER

There are a couple of things to look at. Most network adapter cards use a comm interrupt, usually `COM2 (IRQ3)`. That can usually be changed, but you will also have to change the Novell workstation driver to use the different interrupt. If you are receiving most, but not all of the characters from the comm port, then you may be getting zapped because Netware disables interrupts for long periods of time.

A. From: JOHN RILEY (Lattice)

Yes, a lot of network cards use the serial I/O interrupt lines. This can cause all sorts of problems with either the network software, or in your case, SideTalk (or any serial communications program). What you need to do is to try using a different comm port (`COM2` instead of `COM1` for example). You may be able to see which IRQ line is taken by the network. From there you should be able to find a working serial port.

A. From: RICK SEGAL

We have both Novell and SideTalk. When using a Micom card and the 3c501/3c505 cards, you can't really run SideTalk in the foreground with any degree of luck. You can, however, run `comm2` and the Novell intelligent card and shell. This seems to work okay.

Division With High Bit Set

Q. From: JIM LINDSAY

Is it true???: "`unsigned long a = (unsigned long) b / (unsigned long) ffff`" comes out to be 0??? When I divide by 7fff everything works just fine. What is going on here?

A. From: JOHN MEISSEN (Lattice)

The sign extension occurs because the constant is (by default) of type `int`, which is 16-bits on the 8086. You then `= cast =` it to a long. The cast operation propagates the sign bit. `0xffffL` works because you are specifying the constant to be of type long in the first place. Hence, no sign bit set.

To specify a constant as unsigned you would say `0xffffU`. Anyway, I suppose you mean the cast is to 'unsigned long', so why do the sign extension? I suppose the logic goes:

```
convert to a long
convert to unsigned.
What if you try:
x = ((long)((unsigned)0xffff));
```

Memory Address on a 386

Q. From: GARY MCKEE

I am looking for the address of the 256K of memory that is above the normal 640K memory used by DOS on a Compaq 386.

A. From: BILL ROPER

Try 1024 Kb (although I haven't worked with the machine). The rest of the space below that should be reserved to the BIOS, video, etc.

A. From: GLENN DOBRATZ

According to my 386 Tech Ref manual the 384K of memory above the 640K that DOS uses is located at the high end of the 16MB 386 address space. If you have the Tech Ref, the information you seek is located on pages 3-4. The base address of the 256K that you are permitted to use is `FA0000h` (Yes, 4 zeros). Be forewarned that if you are using `VDISK` as distributed by `COMPAQ`, its first preference is to use this 256K in what ever space you tell it is available.

Continued on next page...

Product Support

...Continued from previous page

Turning Off DOS Interrupts

Q. From: JIM LINDSAY

We would like to turn off interrupts temporarily (while writing some stuff to the screen). I don't see anything in the DOS Technical Reference. Can anyone either, (a) tell me how to do it, or (b) tell me it can't be done? Thanks in advance.

A. From: CLIFF SHARP

I don't know of a standard existing way to turn off interrupts in Lattice, but you could write two functions in assembly language to turn them off and back on again. The assembly language instructions you'll need are CLI (CLear Interrupt flag, disabling interrupts) and STI (SeT Interrupt flag, enabling them). However, You may want to be careful about the amount of time you keep the interrupts shut off.

A. From: SCOTT FISHER

If you are writing in assembly, the CLI instruction will turn off ALL interrupts. A STI instruction will turn interrupts back on. It is NOT a good idea to leave interrupts disabled for any length of time since it could affect the system time clock etc. Also, failing to re-enable interrupts upon exiting your routine will generally cause the system to "hang" — requiring a cold start to get the system operational again. This occurs because the keyboard interface is actually an interrupt driven procedure.

If you are turning off the DOS interrupts to prevent "snow" on the screen, this is not the method to use. You will need to watch the vertical retrace bit in the CRT controller. By updating the screen during a vertical retrace, no snow will appear.

Optimizing For Switch Statements

Q. From: CARL KUCK

In a large model program, I have the following code:

```
switch (unsigned int)
case 0xA000:
    /* do something */
    break; case 0xA123:
    /* do something else */
    break;
.
.
.
default:
    /* do something completely different */
    break;
```

This implements a memory map for a 6800 emulator. The implemented I/O addresses are separate cases in the switch. Would putting the most frequently accessed case first help the performance? Does anyone have any suggestions as to a faster way?

A. From: TOM PRODEHL
(Lattice)

It depends on how many cases you're talking about and how closely spaced together they are.

One option if they are close together is to define separate functions for each case and create an array of functions and calculate an index into the array.

If there are only a few cases and you definitely know the order of more frequent to least, use nested "if then else" "if then else's".

A. From: JOHN MEISSEN
(Lattice)

The compiler should handle the case evaluation in the most efficient order, regardless of how you specify them (note that 'efficient' in this case refers to code generation). If you have a large number of cases, then the order is immaterial since it should generate a branch table that gets indexed by the case value. However, if there are only a few cases, or the values are very far apart, then it will probably generate if-else comparison type logic. It may re-arrange the evaluation order on you, so it may or may not help.

The best bet is to actually code it up, compile it, and disassemble it. If it generated if-else type logic, try re-arranging the order by cut-and-pasting and see if it has any effect.

A. From: DANIEL DOMAN
It often depends on the compiler and the memory model being used but one of the great charms of the C language

switch statement is that it generates very efficient code. If you look at the disasm you will usually see a CMP for bounds checking and then a JMP to the appropriate statements via a jump table e.g. JMP CS:[BX + SOME OTHER OFFSET VALUE]; it's a very efficient call. If your switch statement is really huge it may not be possible to generate calls via a jump table and then indeed you would be better off putting your most likely hits first.

A. From: CARL KUCK

Changing memory models is another interesting way of improving performance. When switching from Large to Data models, I observed a 25% increase in the speed of the program I was working on. Unfortunately, I'm now getting close to running out of code space, but if I can squeak it by, I think staying with D model is worth the speed increase.

A. From: TOM PRODEHL
(Lattice)

Also you might (if you haven't already) use the DS model if you can. The DS model is a submodel of the D model that assumes all pointer arithmetic can be done using offsets. This will speed up execution and may save some code space, too, since more efficient and fewer instruction to perform pointer arithmetic can be used. If you need to perform pointer arithmetic over very large distances, though, it won't work.

A. From: CLIFF SHARP

Another way is using more global variables and fewer passed parameters to routines. Since they get pushed onto the stack and a PUSH takes about a year on an 8088 (three months on an 80286), and since some addressing modes for direct variable access take fewer clocks than addressing the stack frame, this can sometimes pick up the tempo a little.

DOS Re-Entrancy

Q. From: NEIL NEWTON

I have been trying to write memory resident utilities. Using DOS functions sometimes messes them up and sometimes cause no problems. Is there any reasonably good literature/example programs that show how to deal with the re-entrancy problem?

A. From: JONATHAN NIEDFELDT (Lattice)

Unfortunately, you can not do ANY DOS calls if the DOS CRIT flag is set. The way DOS works is simple: it uses its OWN stack internally. That means if you call a DOS call that uses this stack, it will destroy the contents (values) of the old stack (it starts at the beginning each time). That means that DOS will not be able to return from the first DOS call.

Look up int 28h for how to get around DOS CRIT being set constantly.

Q. From: GLENN DOBRATZ

Where does DOS return the address of this 'critical flag'? Does it also encompass the case of knowing if it is safe to do disk I/O? I have been searching, in vain, for some way of knowing whether or not it is safe to do disk I/O during certain interrupts and have not found one, yet PRINT spooler does it all the time.

Short of intercepting int 15 and a host of other things and becoming a mini-operating system, how do you know when it is safe to do disk I/O?

A. From: CLIFF SHARP

There's no distinction, so when the flag says it's okay to do a DOS call, it's okay to do ANY DOS call, even file I/O. I tried this on my last job; one of the guys was doing something with a TSR that had to update a file every so often, and it would die a miserable death by the third update. He knew about the flag, but didn't know about the fact that one has to check the WORD, not the BYTE at the given location; when I modified his code to check the word, it worked, and to my knowledge he's still using the program.

A. From: UWE ROSS

Beware, just because DOS thinks it's safe to do disk I/O doesn't mean it really is. INT 13h is not re-entrant under all circumstances either, and there are some pieces of software that do their disk I/O without using DOS, (Norton's stuff comes to mind). It would behoove all who want to do disk I/O in a TSR to trap int 13 and set a flag when the processor goes there, then clear the flag when it leaves. If the flag is set, don't do disk I/O! (Admittedly 99.9% of the time,

applications do all their disk calls through DOS, but if you happen to be running one that doesn't, the DOS criti-

cal flag isn't worth the cost of the two bytes of memory it consumes!)

Focus on Amiga

Additional Information on the Lattice C Compiler

As a programmer you realize that no product is ever *really* finished. There are always slight improvements to make and new features to include. At Lattice we are always working on our products, but at some point in time the manuals must go to the printer's. During the time the manuals are being printed there are usually some changes made to the product.

So we have included here some suggestions, hints, and miscellaneous information that was not able to be printed in this version of the manual.

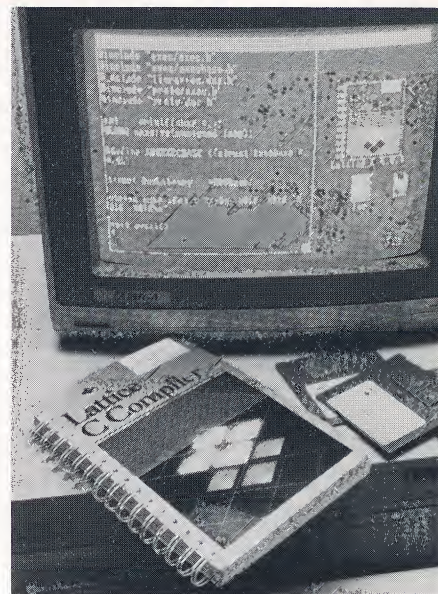
An Extra Disk for the Extra Files

Version 4.0 is now distributed on four Amiga diskettes to accommodate all the last minute additions. The Extras directory on disk 4 contains several useful items from the Lattice BBS including **ConMan**, the console handler used with this package.

Disk 4 also contains the following additional libraries:

lcnb.lib
lcmnb.lib
lcmffpsnb.lib
lcsnb.lib
lcmsnb.lib
lcmffpsnb.lib
amiga.alib — the original amiga.lib (no indexing)

These libraries are compiled with the -bO option and are necessary only for those programs that can't be compiled without -bO. They correspond to the libraries shipped with 3.10 except these now have indexing.



Hints for Installation & Operation

The compiler package is a completely self-contained environment. Disk 1 is a bootable disk that leaves you in the CLI environment with the necessary logical names already assigned. Therefore, on a floppy disk based system no installation is necessary; simply insert disk 1 when the system prompts you for the Workbench disk after the power-on or ctrl-amiga-amiga re-boot sequence (the boot procedure will require both disk 1 and disk 2 to complete).

The Compiler is shipped with AmigaDOS and WorkBench V1.2, and requires V1.2 of KickStart. Also, to provide the maximum amount of free disk space, no fonts or printer drivers are shipped with the compiler. After backing up the disks, copy the appropriate printer driver from the devs/printers directory on

Continued on next page...

C Compiler Info

...Continued from previous page

your WorkBench disk to the devs/printers directory on disk 1.

Workbench (the icon based environment) is not loaded by the boot procedure. This allows the maximum amount of memory for program development. If you want Workbench, simply type the command "loadwb" at the CLI prompt.

We suggest that if you will be using the compiler on a single drive system, you can keep the the number of disk swaps to a minimum by copying the source file to ram and compiling it from there. Of course, the optimum development setup is a 2-megabyte ram expansion. Then you simply copy the compiler executables, library files, and header files to ram disk (see s/install.hd).

If you are using a two-drive system, you may find that the link process is faster if the destination drive is not the same as the drive with the libraries. Using disk 1 for source files, or using the -o option with the driver program 'lc', can significantly reduce the number of disk seeks performed during linking.

Try to use unsigned short integers for array subscripting. The compiler will generate in-line hardware multiplies for address calculations instead of calls to a function to do long integer multiplies. Although highly optimized, the long integer multiply routine is still over 3 times longer than the single in-line multiply instruction.

Aborting a compile (or any program created with the default break-character handler in this version) causes a requester to appear allowing the user to continue or abort. Under V1.2 of AmigaDOS these options may be selected via the left-Amiga key in conjunction with either the 'v' or the 'b' key. Left-Amiga-v continues, while left-Amiga-b aborts.

A Recoverable RAM Disk driver from ASDG is available on the Lattice BBS. Copyright restrictions prevent us from including the RRD with the compiler. However, we highly recommend you take the effort to obtain it. If you have a large expansion memory then the recov-

erable RAM disk driver can be used to implement a compiler environment that can be recovered if the machine crashes.

If you are creating a Terminate and Stay Resident program by using cback.o, you can now tell LC to automatically link in cback.o instead of c.o by using the -t option on LC. For example to compile popcli you could do:
LC -t -Lcdnv + popsup.a -dTINY = 1
_main.c popcli.c

When using the background task startup routine cback.o the linker may generate the following error message:

Error 515: An ALV was generated pointing to data symbol

This message should be a Warning message, and should display the symbol name XCEXIT.

The message itself is normal, and is a consequence of the method used by cback.o to establish the program as a background task. In order to generate relocation information that would be relevant to data that might be copied to another part of memory, the exit code is assembled in a data section. Normally a C program would never reference something in a data section as a function.

Finally, to assist in debugging with Metascope, you can invoke the ADDSYM option of blink by using the -La option on LC.

Undocumented Compiler Options

- ce Disables printing the source line that the error was found in along with the error message.
- cf Causes the compiler to insure that prototypes exist for all functions in the source file.
- c+ Disables the compiler warning message about using structures as parameters in function calls.

Product Support

Following are the most frequently asked questions concerning the Lattice AmigaDOS C Compiler. If you have any questions on any Lattice product, please feel free to write, call, send a message on our FAX, or utilize our Bulletin Board.

Q. How do I use the short integers?

A. The -w option on LC (and LC1) is used to tell the compiler to use short (16-bit) integers. When you use this option, all integers are passed as 2 bytes on the stack and are calculated with the short form of the 68000 instructions. This is a performance increase over the standard long integer default but it does come with some pitfalls. In particular if you fail to declare a subroutine that returns a pointer, it will instead only return a 16 bit short which is not large enough to hold a pointer. Also if you pass a short integer to a function that expects a long integer or a pointer, then the parameters will not match up. This is especially nasty when passing NULL because exec/types.h #defines NULL as 0.

The compiler has a couple of features to help with the -w option. If you use -w and fail to declare a routine but then use the result of a routine as a pointer the compiler will issue warning message number 101 "short value converted to a pointer". To solve the problem of parameter mismatches, use the -cf option on LC1. (see below)

Q. How do I use the -cf option?

A. To solve the problem of parameter mismatches, the -cf option on LC and LC1 will cause the compiler to issue warning message number 96 "No function prototype for function" whenever you call a function without a prototype. This option is very useful to find bugs in your program even before you attempt to run it. In general if you can compile with the -cf option and get no warnings, you will not run into parameter mismatch errors.

The only potential area of problem comes with functions that take a variable number of parameters such as `forkl`, `forkv`, `printf`, `sprintf` and `fprintf`. For these, you simply need to make sure that you are passing the appropriate parameters.

Q. What is a prototype and how can I define them?

A prototype is a declaration of a function that tells the compiler what the types of parameters are. For example `Open()` takes a character string and a long returning a `BPTR` and is declared as:

```
BPTR Open(char *, long);
```

We have supplied prototypes for all Amiga functions and the Lattice library functions. You need to `#include` the file for the functions that you are calling. If you are using `exec` functions, you should `#include <proto/exec.h>`. For dos functions, `#include <proto/dos.h>`, etc. The Lattice functions are found in:

<code>stdlib.h</code>	for general functions
<code>stdio.h</code>	level 2 i/o
<code>ios1.h</code>	level 1 i/o
<code>dos.h</code>	operating system interface functions
<code>string.h</code>	string functions

For your own program, we suggest either putting prototypes for all functions at the top of the source file or in a `#include` file if your program consists of more than one source file. In general it is good practice to have prototypes for everything as it lets the compiler check every time you compile your code. They also serve as documentation to tell you what parameters a function takes.

Q. My program used to compile, but now I get error message #510 from the linker. What is wrong and how can I fix it?

A. With the 4.0 release, we have changed the default for all data from using the non `-b` relative addressing to `-b` relative. This means that while before the code used a 32 bit address that was fixed up by the Amiga loader when your program ran, we now use a 16 bit value assigned by the linker as an offset from a global base data register (now `A4`). This means that the code can use shorter

(and faster) instructions and there is less relocation information in the load file.

The side effect of this is that ALL data must fit into this one global data section. There are 3 exceptions to this rule:

1. Items destined for chip with the `-c` option on `LC2`
2. Other modules compiled with the `-b0` option
3. References to `CUSTOM` or the `CIA` defines.

You have several ways around this limitation:

1. Compile everything with `-b0`. This gives larger code but is the least troublesome thing to do.
2. Put all accesses to the trouble items into a separate module and compile only that module with `-b0` (yes you can mix code that has been compiled with `-b1` and `-b0` in the same program)
3. Use a level of indirection in the declarations to access the problem variables. A good example of this can be found in `POPCLI` in the examples directory for accessing custom. This is done as:

```
extern struct Custom custom;
struct Custom *customptr =
&custom;
#define custom (*customptr)
```

By doing this, there is no need to change the code at all.

Q. How can I get rid of that window when I run from workbench?

A. The window is automatically opened by the code for `_main` when it detects that you ran from workbench. This is to allow you to write a program such as 'Hello world' which does a `printf` to workbench for which there is no output console. To get rid of this feature, you must recompile `_main.c` (provided on disk 3 in the source directory) with the `-dTINY` option:

```
LC -dTINY _main.c
```

Note that once you do this, you cannot use default standard I/O (`printf`, `puts`, `gets`, . . .) because it will crash if you are invoked from workbench.

Q. When I link my program with `cback`, it either crashes or opens up 2

windows from workbench. What is wrong?

A. Your program is requesting standard I/O through `_main`. Since a background process has to respect special rules for its output, you need to get rid of the default standard I/O set up for it. This can be done by recompiling `_main` with the `-dTINY` option as above.

Q. When I link my program with `cback.o`, I get error message 515 from the linker. What is wrong?

A. This is a normal message. The diagnostic is there to let you know when an attempt is made to call something in the data section. However, `cback.o` must work its magic for `XCEXIT` by creating a short stub routine in the data section. When you call `exit`, `exit` or `XCEXIT`, the linker eventually resolves your code to this stub in the data section, hence the message.

For programs not linked with `cback.o` you should pay attention to the message as it indicates a potential problem.

Q. In my very large program, I am comparing two function pointers to the same function but they don't compare equal. What is wrong with the compiler?

A. There is nothing wrong with the compiler, BUT because `-r` is now the default, the compiler uses PC relative addressing for all subroutines. If your code takes the address of a function it will use a 16 bit PC relative offset to access the function. If the target of the branch is greater than 32K away, then the linker will have to create an ALV stub to access the function. If you pass this address to another function then it may call through this address without problem. However IF you attempt to compare the address to another function address, it MIGHT not be equal because the linker may have resolved the addresses to different stubs.

There are 3 potential solutions to this problem:

1. Compile everything with the `-r0` option. (or only those modules that take the address of the functions in question).

Continued on next page...

Product Support

...Continued from previous page

2. Arrange the modules so that all modules affected are close enough that the linker does not create an ALV for the routine in question. You can detect an ALV by looking at the cross reference section of the linker map for the symbol. If a reference appears in parens then it is an ALV created by the linker.
3. Call a function that does the comparison of the two pointers taking into account ALV's. Such a function would look like:

```
int compfunc(f1, f2)
char *f1, f2;
if (f1 == f2) return(1);
/* Prevent blowing up for odd values */
if (((long)f1 & 1) ((long)f2 & 1)) return(0);
if (*(short *)f1 == 0x4ef9) f1 = *(char
**)(f1 + 2);
if (*(short *)f2 == 0x4ef9) f2 = *(char
**)(f2 + 2);
return (f1 == f2);
```

You can then code:

```
if(compfunc(p1, p2))
where you would have previously
coded
if(p1 == p2)
```

When you wish to compare the two function pointers.

Q. You advertised 1294 Dhrystones in the ads, but I can only get 900 (or 980). What gives?

A. Several things are going on. The other two important things to note are that the Dhrystones fail to `#include <string.h>` even though it uses string functions. The other is that we compiled the Dhrystones with the `-w` and `-v` option to use the same options that our competitor had been advertising with. To get the 1294, you need to:

1. Apply the enclosed patches to the compiler
2. add a `#include <string.h>` to the Dhrystones source code. (Note if you had compiled it with `-cf` you would see the warnings for the failure to declare the routines)
3. compile it with:

LC -L -w -v Dhrystones

What this shows is that our compiler can produce a wide range of code with varying levels of support and safety. Our default is safety and convenience for you the programmer, yet we give the switches for you to tell the compiler to let you go out on your own.

Q. If I `#include <proto/cstrings.h>` I get an undefined reference to `CStringBase` from the linker and my program crashes when I run it.

A. `proto/cstrings.h` has been provided for the few developers who have a copy of the `cstrings` library provided by Commodore. This is a library that was included in earlier versions of the operating system but are no longer distributed. If you don't know that you have it and why you have it then you don't need `proto/cstrings.h`. The compiler libraries provide all the necessary string functions for normal C programs; `cstrings` is simply another flavor of handling strings.

Q. What are all these libraries on disk 4 with the 'nb' in the name?

Q. Because we change the compiler default to `-b`, we elected to provide a set of libraries compiled without the option so that you could write code that does not use the option or even wants to set up the base pointer. For normal usage you will not need these libraries.

Q. My program worked under 3.10 but seems to crash under 4.0. Have I found a compiler problem?

A. Certainly with a new product it is possible, but it is more likely that the new code generated by the compiler has brought out a latent bug in the program. We have found several programs that just happened to work under 3.10 but when recompiled under 4.0 failed to run (an earlier version of DU is a good example) but in all cases compiling with the `-cf` option showed up a routine which wasn't prototyped and caused the problem. To be fair, we have found the opposite to be true, programs that didn't run under 3.10 (trek73 is one example) now run under 4.0 without problems.

We recommend that you first compile your program with the `-cf` option and eliminate all messages first. This hopefully will eliminate any bugs. If your problem still persists, feel free to contact Lattice Technical support. As you can see, we are serious about supporting and fixing any problems in the product and will be more than happy to help out.

Focus on RPG

Two New Programs Strengthen RPG II Product Line

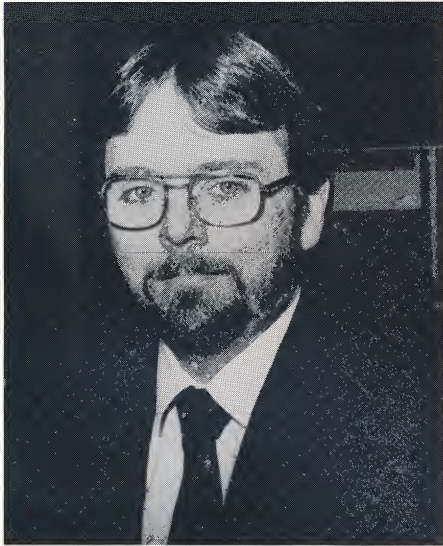
The RPG II compiler and utilities are top selling products at Lattice," said Wayne Nartker. "We expect this to continue as PCs are incorporated into 3X shops at an increasing rate. Our PC-based compiler offers a very cost-efficient way for these companies to recover the 3X system's resources by recompiling and running stand-alone ap-

plications on the PCs. Therefore we are taking steps to ensure that we have a constantly-improving product available to users from a growing number of outlets."

Currently Lattice is finalizing an "Authorized Reseller Program" designed to increase support for consultants, VARs and dealers who distribute the

product and provide the first level of support to end users.

John Robinson will administer the new RPG II reseller program which offers substantial discounts and incentives to resellers of the RPG II product line. There is a good chance many of you already know John. He joined us in November 1987 — bringing with him more than seven years' experience providing VAR and corporate sales support as well as customer support. Before coming to Lattice, he worked for companies including First Computer Corporation, and ComputerLand.



John Robinson brings to Lattice the experience and qualifications to help you take advantage of Lattice's VAR Support program. Call him today for complete details.

Under the new program, when you become a registered reseller, you receive our standard discount on all Lattice RPG II products plus additional discounts based on your quarterly sales volume. You can also take advantage of Lattice's 30-day credit policy and Lattice's 30-day money-back guarantee for your customers. Other proposed terms of the agreement include free demo units, lead referral from Lattice's extensive advertising schedule, and free sales literature.

"Lattice will continue to offer technical support for the products," said Robinson. "However, this program makes it easier for users to obtain pre-sales support from a knowledgeable consultant or VAR. In addition, most end-users are more comfortable obtaining the front-line support from a nearby

consultant or dealer before calling Lattice."

In addition to creating a reseller program, Lattice is also accelerating RPG II product development.

Robert Hansen, Lattice Vice President of Engineering, has expanded the RPG II product development staff. And he is committed to transferring many of the user requests received by the technical support group into product enhancements. According to Hansen, "The next release will support color monitors, key mapping, and improved network capabilities. We are currently working on adding a DFU (Data File Utility), an OCL processor, and an OS/2 version. We will also continue to examine language extensions for programming a PC with features not available on a 3X system."

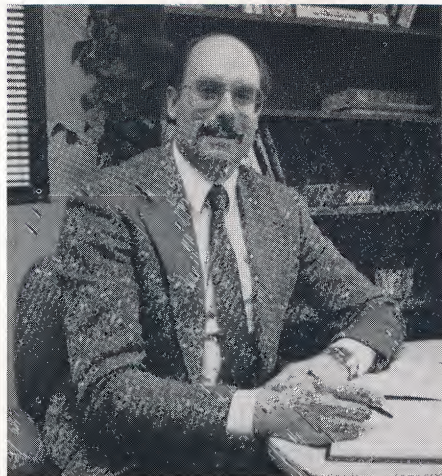
Our current version of the compiler is RPG II compatible with System 36, Version 5. We also offer a complete programming environment consisting of the RPG II Compiler, Source Entry Utility, Screen Design Aid, and Sort/Merge Utility. Unlike other PC-based RPG II compilers, there is no run-time system or license required for programs created with the Lattice RPG II compiler.

"When our editing and design utilities are combined with a PC's single user,

single task environment, the program development cycle is greatly reduced," said Hansen. "Whether your programs will be run on a System 3X or a PC, you will notice an increase in your efficiency by writing, compiling and testing the programs on the PC."

For information on the Lattice RPG II Development System or information on becoming an authorized reseller, call us at (312) 916-1600 or send your request to: John Robinson, Lattice, Inc., 2500 South Highland Avenue, Lombard, IL 60148.

"...whether your programs will run on a 3X System or a PC, you'll notice an increase in your efficiency by writing, compiling, and testing your programs on the PC..."



Product Support

Following are some frequently asked questions concerning the Lattice RPG II products. If you have any questions on any Lattice product, please feel free to write, call, send a message on our FAX, or utilize our Bulletin Board.

Q. SEU automatically compresses my source files. How can I keep files from being saved in a compressed mode?

A. When you go to save your source file with the SAVE command (F3), the Lattice SEU will automatically convert multiple blank spaces into tab characters to save disk space. You can keep this from happening by adding a "/n" option to your SAVE command. All spaces will be saved as blanks and you will have a standard ASCII file without tabs. See page 3-14 in your SEU manual for more information.

Q. How do I specify packed decimal data with the ICREAT utility?

A. Before answering this specific question about ICREAT, first it is necessary to clear up some confusion about ICREAT and the building of index files.

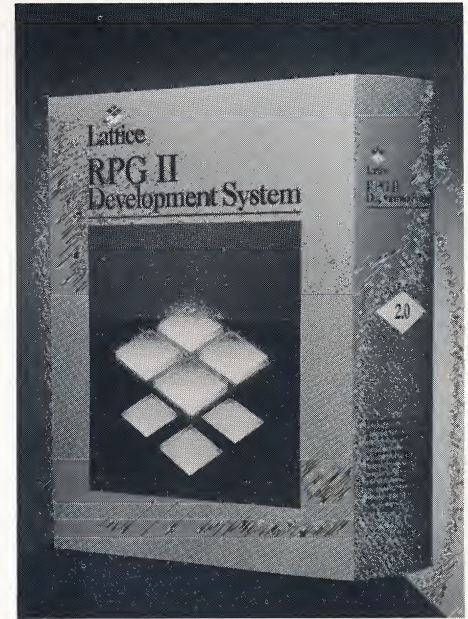
Index files can be built through RPG Output specifications or by using the Lattice Sort/Merge utility ICREAT. See the example source INIT.RPG on disk 3 of your RPG compiler for more information on using an RPG program to build index files.

The Lattice index file structure is based on the dBASE III file standard. There are two files that make up the index file - a file that contains the data records and has the file extension .DBF and a file that contains the keys and has the file extension .NDX.

Each file starts with a header record. This record is used by dBASE III and the Lattice ISAM File Utilities and contains information about the fields and data in the file.

The difference between building an index file with an RPG program and ICREAT is in the header record. An RPG program will build an index file with generic field information, while ICREAT allows you to build an index file with specific field information.

When using an index file in an RPG program, the only header information that affects the program is the record length and key specifications - length and type. The number of fields, specific names, field types, and decimal position information in the header record are merely extra documentation.



The value in specific header information is for dBASE III or the ISHOW or INDEX utility. For example, with specific header information you can build alternate indexes with the INDEX utility.

Therefore, to answer the specific question on building an index with pack data, the only thing you need to specify that affects the file is the actual data or field length in the file record. For example, if the data is contained in 3 bytes on the record, specify the field length in ICREAT as 3. The field type and decimal position will not matter.

NOTE: While there is no difference in file structure compatibility between Lattice RPG programs and dBASE III, there is the potential for data incompatibilities. dBASE III can not read packed decimal or binary data.

Current Product Version Numbers

Software for the IBM PC and Compatibles

Version

Lattice MS-DOS C Compiler	3.22
Lattice OS/2 C Compiler Upgrade	5.00.07
C Cross Reference Generator (CXREF)	1.03
C-FOOD Smorgasbord™	3.20
Code Sifter	1.20
C-SPRITE™ Debugger	2.02
Curses Screen Library	1.05
dBC III™ Library for Lattice C	2.00
dBC III™ Plus for Lattice C	1.01
Lattice Make Utility (LMK™)	2.20
Lattice Screen Editor (LSE™)	1.00
PANEL Plus Forms Manager	1.10
Scientific Subroutine Package for PC's (SSP/PC)	1.10
HighStyle	1.00
SecretDisk® II	1.00
SideTalk	1.00
Text Management Utilities (TMU™)	2.10

RPG Software for the IBM PC and Compatibles

Version

Lattice RPG II Compiler	2.01
Lattice Sort/Merge (LSM™)	2.00
Screen Design Aid (SDA)	2.01
Source Entry Utility (SEU)	2.00

C Cross Compilers for MS-DOS Targets

Version

Lattice Apollo to MS-DOS C Cross Compiler	3.11
Lattice Pyramid to MS-DOS C Cross Compiler	3.20
Lattice SUN to MS-DOS C Cross Compiler	3.11
Lattice VAX/BSD to MS-DOS C Cross Compiler	3.11
Lattice VAX/SYS V to MS-DOS C Cross Compiler	3.11
Lattice VAX/VMS to MS-DOS C Cross Compiler	3.24
Plink86plus Cross Linker	2.12

C Cross Compilers for Other than MS-DOS Targets

Version

Lattice MS-DOS to AmigaDOS C Cross Compiler	3.10
Lattice MS-DOS to NEC 78312/78310 Cross Development System	1.00
Lattice MS-DOS to 68000 C Cross Compiler	3.04
Lattice MS-DOS to Z80 C Cross Compiler	3.04

Software for the AMIGA

Version

Lattice AmigaDOS C Compiler	4.01
dBC III™ Library	2.00
Lattice Make Utility (LMK™)	1.00
Lattice Screen Editor (LSE™)	1.10
MacLibrary™	1.01
PANEL Forms Manager	6.20
Text Management Utilities (TMU™)	1.01
Unicalc® Spreadsheet	1.10

Software for the Atari

Version

Lattice Atari C Compiler	3.04
--------------------------------	------

This list is included to keep you informed of our current product version numbers. Refer to the history portion of your Lattice Update Card to determine if your product has changed versions.

Compiler Companion Now Offered

If you do any programming on your Amiga, we're sure you'll appreciate our new **Compiler Companion** which consists of ten proven utilities designed to enhance your productivity.

The Compiler Companion features EXTRACT and BUILD, which extract file names from a directory and build a command file from the list; CXREF, which generates a cross-reference listing of C language source files; DIFF, which compares files and reports the differences; FILES, which locates files by specified attributes; GREP, which matches character string patterns within a file; LMK, the Lattice Make Utility which automates the management of files based on dependencies and rules; SPLAT, which searches and replaces within one or more files; TOUCH, which updates the date and time stamps of individual files; and WC, which counts the number of words within a file. The price for all this is only \$100.00.

"This collection of utilities provides a complete programming environment for the Amiga regardless of the programming language used," said Wayne Nartker. "All the utilities but CXREF are totally independent of the programming language used. This means that BASIC,

MODULA-2, FORTRAN, PASCAL, and ASSEMBLER programmers as well as C programmers will benefit from this product.

"You may recognize that the programs in the Compiler Companion replace the CXREF, LMK and Text Management Utilities that we sold separately in the past," continued Wayne. "However, we have re-compiled them using Version 4.0 of our AmigaDOS C Compiler so they require less memory and execute much faster. In addition, we have created a single new manual that covers each utility in detail. And at \$100.00, the Compiler Companion is substantially lower than purchasing the individual products."

You can get the Compiler Companion from dealers and distributors worldwide or directly from Lattice. If you want additional information, call our sales department at (312) 916-1600.



Lattice

Lattice, Incorporated
2500 South Highland Avenue
Lombard, Illinois 60148

Subsidiary of SAS Institute Inc.

BULK RATE

U.S. Postage
PAID

Glen Ellyn, IL
Permit No. 597

Theodor H. Nelson

8480 Fredericksburg #138
San Antonio, TX 78229

Return Address Requested